Appl. Math. Lett. Vol. 6, No. 2, pp. 17–21, 1993
Printed in Great Britain

0893-9659/93 $6.00 + 0.00
Pergamon Press Ltd

UNSTRUCTURED RECTANGULAR MESH GENERATION FOR ADAPTIVE TWO-DIMENSIONAL CONSERVATIVE SCHEMES

M. A. KELMANSON
Department of Applied Mathematical Studies
University of Leeds, Leeds LS2 9JT, England

(Received June 1992; accepted July 1992)

Abstract—A scheme is presented for the automatic generation of unstructured rectangular meshes suitable for the integration of dynamically adaptive, time dependent Eulerian codes. Our approach generates meshes quite different from those in the existing literature, which are primarily based upon block structured or grid embedding algorithms, and therefore have the inherent disadvantage that they inevitably place fine mesh cells in regions somewhat larger than actually necessary. Our meshes do not suffer this drawback: they possess fine cells *only where necessary*, and not in blocks extending into regions of relative 'inactivity.' Results are given for the particular cases of the initial configurations of rectangular and circular penetrators striking an oblique plate.

1. INTRODUCTION

The problem of discretising an n-dimensional ($n = 1, 2, 3$) domain has, for a long time, been a fundamental preprocessing calculation in almost all areas of scientific and engineering analysis. It is a first requirement in being able to solve a discrete form of a continuous system of partial differential equations via a particular numerical scheme. *Regular* meshes have traditionally been used to solve the discrete problem: these suffer the dramatically wasteful setback that, if very fine resolution is required in a certain part of the mesh, it is necessarily enforced throughout the mesh. In recent years, considerable effort has been directed towards *adaptive* meshes, so-called because they adapt themselves (on the basis of either physical or geometrical information) in such a way as to create fine mesh cells in only those regions where high resolution is required, and coarse cells elsewhere. The advantage of this approach is twofold, in that

(i) most obviously, storage requirements may be considerably reduced, and
(ii) the decreased number of cells will lead to faster solution integration in time-dependent codes.

For further introductory material, and a brief review of recent literature in this field, see [1].

We have found that much existing literature utilises *block structured* approaches, as in [2] or *embedded grid* techniques, as in [3–5]. These types of adaption schemes essentially superimpose locally refined rectangular meshes, on top of coarser ones, in those regions where increased resolution is required. They share the common properties that

(i) the meshes so-produced are somehow *structured*, and
(ii) they inevitably produce refined cells in regions outside those where the finest resolution is required.

The motivation behind the present work is to generate *automatically* a computational mesh which is

(i) devoid of any artificial restrictions,
(ii) truly *unstructured*, and
(iii) 'finely tuned' so that it refines cells only where the *finest resolution is required*.

This work was undertaken while the author was in receipt of finanacial support from the Defence Research Agency, Fort Halstead, Sevenoaks, Kent, England under agreement D/ER1/9/4/2062/142/RARDE.

The flexibility afforded by these properties creates a computational mesh permitting the implementation of very general solvers which are transparent to the details of any particular problem. Discussion will now be restricted to the case of *rectangular* meshes as employed by two-dimensional finite difference schemes and, in particular, conservative schemes.

2. ADAPTIVE MESH GENERATION

We have already mentioned the slightly restrictive nature of the block structured approaches above. We remark that much (rectangular mesh) adaptive literature incorporates the following completely artificial restraint, which is enforced purely to simplify implementation aspects: *each cell should be exactly one half (or double) the size of its immediate neighbours in the mesh.* A brief consideration of this point will reveal just how 'polluting' its enforcement can be, for if one cell is refined, all of its neighbours, and all of its neighbours' neighbours, etc., need to be refined. We omit this restriction in our mesh generator.

As an example, we have fitted a static, adaptive, multi-level mesh to the configuration of a rectangular (and circular) penetrator impacting an oblique target. The mesh generator is used as a preprocessor for an Eulerian finite difference code, developed by Maunder [6], for the *dynamically* adaptive solution of time dependent impact phenomena. An initial coarse mesh is interactively selected, as are all the physical parameters of the system, e.g., penetrator type, size and yaw, target obliquity and thickness, and the maximum required number, L say, of levels of adaption. For example, if a 2-by-2 coarse mesh is initially selected, then the finest cells will correspond to a regular mesh of 2^{L+1}-by-2^{L+1} cells. The storage savings achieved via this approach are appreciable, and are discussed below.

The adaption process is briefly described as follows. Each cell in the original coarse mesh is traversed in a preselected order and the (physical) information pertaining to that cell is appended onto a *linked list.* (We remark at this stage that the C programming language was used in order to implement the required concepts of *data structures, pointers, linked lists* and *dynamic memory allocation.*) Next, the coarse cell so-produced is inspected to see if it requires refinement; the *adaption criterion* necessary for this decision is fixed *a priori.* We chose to refine whenever a cell contained more than one fluid (note that our preprocessor examines a three-fluid—air/penetrator/target—system). In this way, refinement will accumulate the finest cells around material interfaces, which is exactly where we wish them to be. If refinement is necessary, the old coarse cell is symmetrically quartered and discarded; the information pertaining to the four new cells is then 'woven' into the linked list using the process of *dynamic memory allocation.* The advantage of this approach is, of course, that we don't need to know *a priori* the number of cells we shall ultimately require. And so it proceeds, down to the finest permissible level, before passing back on to coarse cells.

The above description of the adaptive mesh generator is necessarily oversimplified in a paper of this size. We remark that its implementation required a substantial initial programming commitment, but that it can now be used as a preprocessor for a variety of problems, simply by changing the module in which the penetrator and target geometries are specified. The reader is referred to [7] for further details, and to [6] for a discussion of the subsequent *dynamic* adaption scheme employed in the study of time-dependent impact phenomena.

Finally, we note that the recently-published, flexible, multiblock approach of Evans *et al.* [8], used for compressible flow computations, differs from the present work insofar as

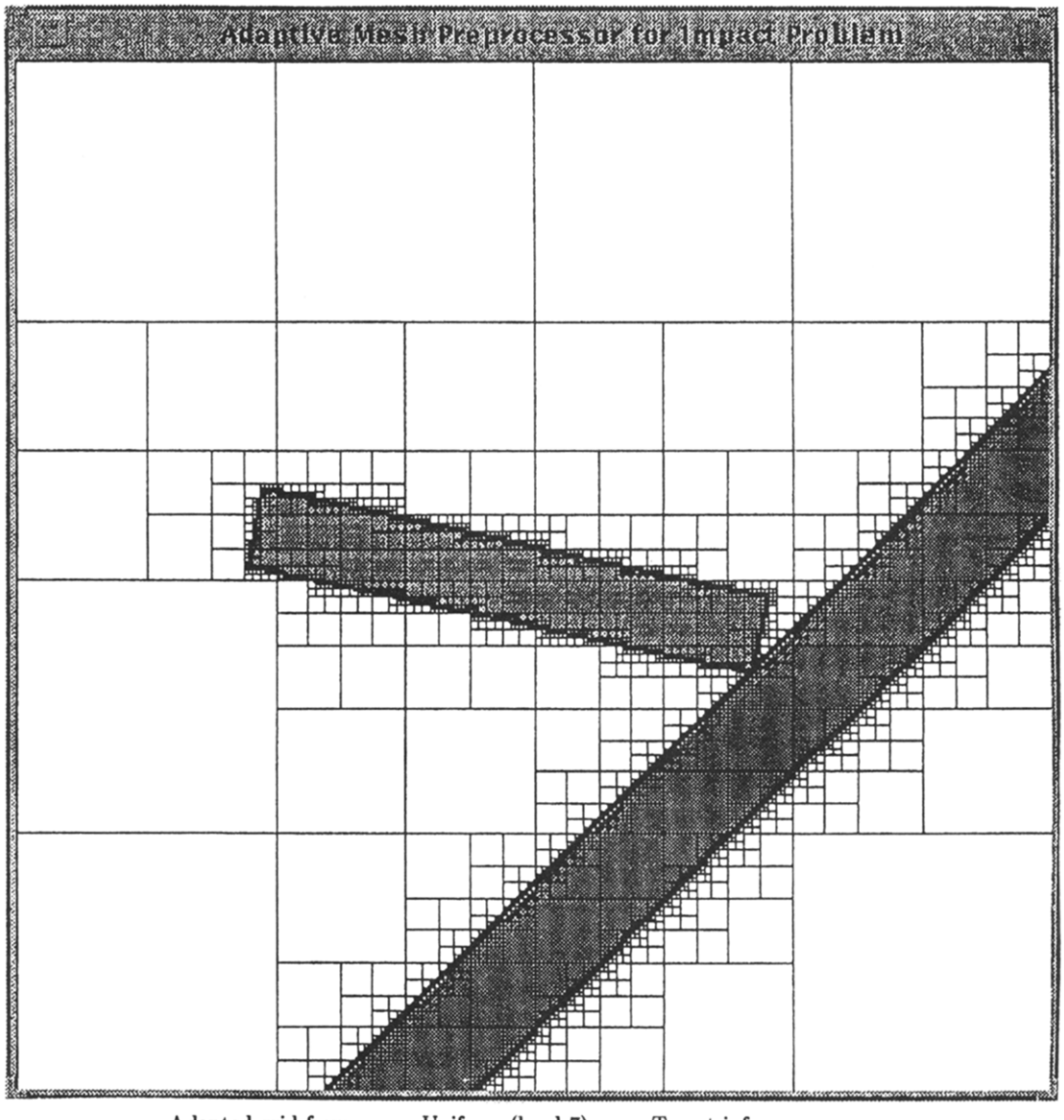
(i) it represents a compromise between a globally structured mesh and a locally connected unstructured mesh, and

(ii) it utilises the data structure known as the *quadtree*, rather than the linked list approach.

The mesh produced in this paper is globally unstructured.

3. RESULTS AND DISCUSSION

In Figures 1 (and 2) we present 7-level adaptions, of a 2-by-2 initial mesh, for a rectangular (and circular) penetrator impacting an oblique plate. Clearly visible are the completely unstructured forms of the meshes so produced, as well as the efficient (i.e., non wasteful) distribution of the finest cells. Note too that the fluid interfaces are represented by fractal-like curves, since the



Adapted grid from	Uniform (level 7)	Target info:
2 by 2 coarse mesh	grid has 256 by 256	Obliquity = 45.000
level 0 = 0 cells	= 65536 cells ==>	Thickness = 1.000
level 1 = 7 cells		
level 2 = 13 cells	storage saving	Physical domain:
level 3 = 33 cells	ratio = 22.843	0.00 <= x <= 10.00
level 4 = 116 cells		0.00 <= y <= 10.00
level 5 = 235 cells	Penetrator info:	
level 6 = 485 cells	Length = 5.000	
level 7 = 1980 cells	Width = 0.750	
(including 993 mixed)	Yaw = 12.000	
Total = 2869 cells		

Figure 1. Adaptive mesh for rectangular penetrator and oblique plate.

meshing procedure makes no attempt to produce boundary-fitted coordinates. The effect, on the solution error, of this fractal representation of the fluid interfaces will form the basis of future investigation. The figures also show information pertaining to the distribution of cells in the processed meshes, namely

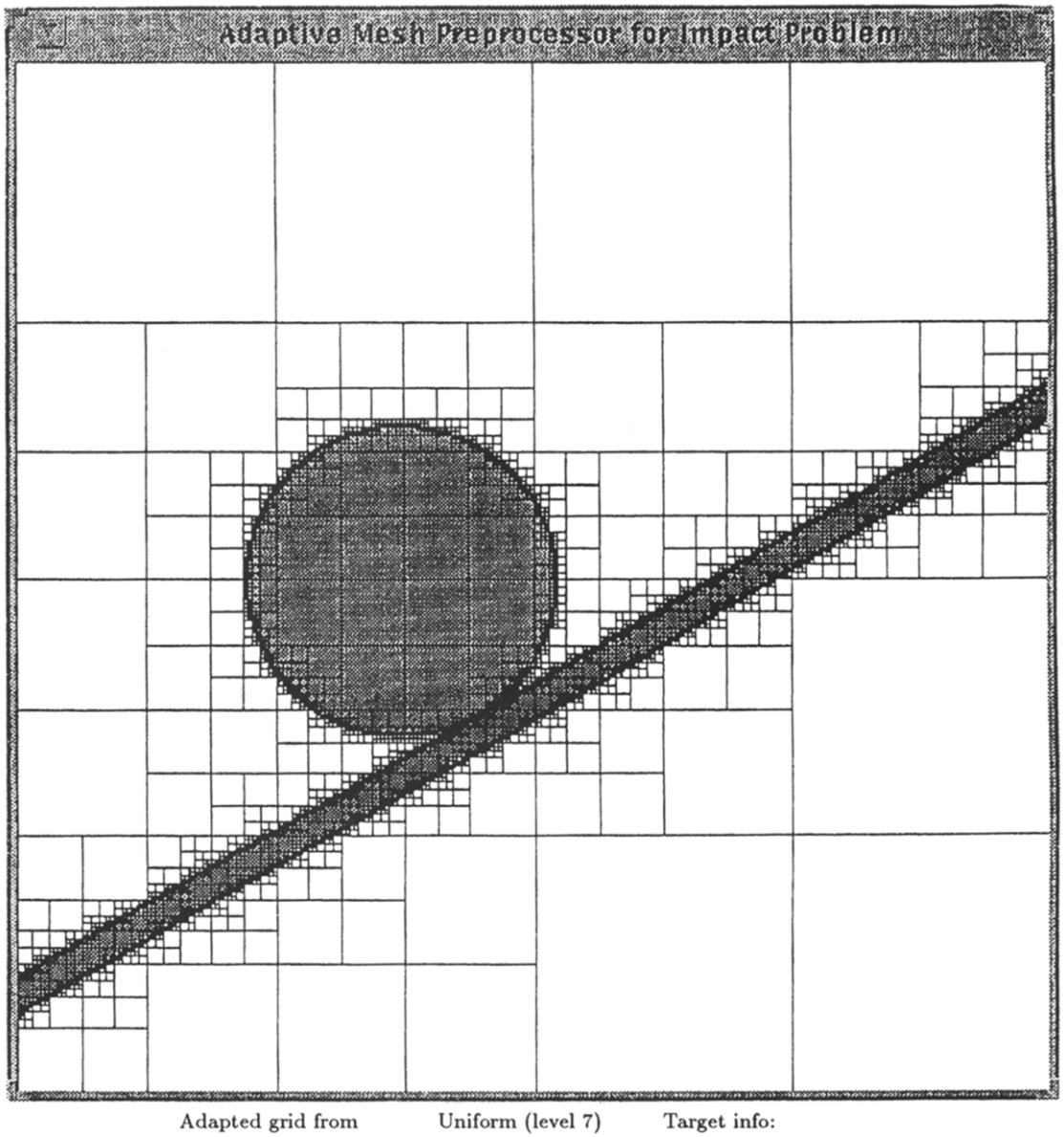


Figure 2. Adaptive mesh for circular penetrator and oblique plate.

(i) the number of cells at each level of the final mesh,
(ii) the number of cells which are flagged for further refinement, and
(iii) the storage-saving ratio based upon an equivalent regular mesh at the finest level.

All figures demonstrate clearly the ability of the preprocessor to refine *automatically* around the fluid interfaces.

Note that, for each figure, approximately 70 percent of all the cells are on the finest level, yet these cells actually cover a substantially smaller percentage of the physical domain. That is, the mesh is highly concentrated around those areas where 'changes' are to expected in the numerical solution of the time-dependent problem. The 'storage-saving ratio' shows how many times more storage would be required if the finest resolution were extended over the whole mesh: it doubles (approximately) as the maximum permitted number of levels is increased. In time-dependent *dynamic* adaption, it is found that, as adaption proceeds, the finest-level cells account for over 90 percent of the mesh, whereas the spatial proportion of these cells does not rise to much above 10 percent of the overall physical domain. Further details of the dynamic process are to appear in a paper by Maunder [9].

References

1. M.A. Kelmanson, Dynamically adaptive rectangular meshes, Report D/ER1/9/4/2062/142, No. 3, RARDE, (May 1991).
2. D.P. Young, R.G. Melvin, M.B. Bieterman, F.T. Johnson, S.S. Samant and J.E. Bussoletti, A locally refined rectangular grid finite element method: Application to computational dynamics and computational physics, *J. Comput. Phys.* **92**, 1–66 (1991).
3. M.J. Berger and J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, *J. Comput. Phys.* **53**, 484–512 (1984).
4. M.J. Berger and P. Colella, Locally adaptive mesh refinement for shock hydrodynamics, *J. Comput. Phys.* **82**, 64–84 (1989).
5. P. Coelho, J.C.F. Pereira and M.G. Carvalho, Calculation of laminar recirculating flows using a local non-staggered grid refinememt system, *Int. J. Num. Meth. Fluids* **12**, 535–557 (1991).
6. S.B. Maunder, A dynamically adaptive 2-D Eulerian impact code utilising unstructured rectangular meshes, Report D/ER1/9/4/2062/147, No. 3, RARDE, (December 1991).
7. M.A. Kelmanson, A menu-driven preprocessor for unstructured rectangular mesh generation in adaptive 2-D Eulerian impact codes, Report D/ER1/9/4/2062/142, No. 4, RARDE, (December 1991).
8. A. Evans, M.J. Marchant, J. Szmelter and N.P. Weatherill, Adaptivity for compressible flow computations using point embedding on 2-D structured multiblock schemes, *Int. J. Num. Meth. Engng.* **32**, 895–919 (1991).
9. S.B. Maunder, Dynamically adaptive 2-D impact mechanics, (1993), (to appear).